SAM 奇怪的补充

KF.Dong

SAM

- 只能识别一个串**所有后缀**的**自动机**；

- 识别的定义：从自动机的源开始，依此读入一个字符然后沿着自动机的边转移，若到达的状态是结束状态，则此串可以被识别。

关于定义和约定

- 如果未说明，大写S表示原串，Fac表示S的所有连续子串，Suf表示S的所有后缀，Suffix(a)表示S从a开始的后缀，S[l,r)表示S的一个子串
- 对于两个字符串a、b，定义ab为将b接在a之后的新的字符串
- ST(s)表示在S的自动机上从起点出发，读入一个字符串s后，在自动机S上到达的状态(state)
- 根据后缀自动机的定义，若ST(s)∈end(结束状态)，则s为S的一个后缀；

一些结论

- 若s不是S的一个子串，那么ST(s)=NULL；因为s在后面加上任何字符都不可能是S的后缀，所以我们不需要记录这个状态。
- 因此当且仅当ST(s)!=NULL时，s是S的一个子串

- 考虑从ST(a)开始，能识别哪些串，即有哪些x，使得ax∈Suffix。显然x∈Suffix，故ST(a)能识别的串还是一些后缀，记Reg(ST(a))={x| ax∈Suffix}

分析

- 由于Reg()是一些后缀的集合，即
  Reg(ST(a))={Suffix(r1),Suffix(r2)…,Suffix(rn)}
- 不妨记Right’(a)={r1,r2……，rn}，也就是Right’(a)和Reg(ST(a))是对应的，可以看出Right’(a)就是a在S中出现的所有结束位置的集合
- 那么若Right’(a)=Right’(b),则ST(a)=ST(b)，这两个状态的所有后继状态完全一样；
- 所以在最简状态SAM中，每一个状态s满足:所有Right’(a)相同的串的转移ST(a)=s，以及所有ST(a)=s的串a，Right’(a)相同，不妨记Right(s)=Right’(a)

分析

- 接下来，考虑一个状态s=ST(a)中的字符串a在原串S中的分布；

- 首先，由于Right’(a)是axi∈Suffix的xi 的起始位置ri的集合，例：
- aaabbaaabd 中

  Right’(aaab)={4,9}，对于一个状态s，设Right(s)={r1…rn},那么如果知道了a串的长度len，就可以确定a在原串S中是[ri-len,ri) (1<=i<=n)，容易看出对于一个len，串a有且只有一个，但是有些len是不合法的，比如s=ST(aaab),len=1时，a串是”b”,但是Right’(a)={3,4,9}，那么ST(a)!=s

- 显然，对于一个状态s，若长度l<=r所对应的al,ar满足ST(al)=s,ST(ar)=s,那么对于任意l<=x<=r,ST(ax)=s,不妨设s所有合法的长度集合为[Min(s),Max(s)]

- 由于Right‘(a)是串a出现的结束位置的集合，那么显然对于两个串a,b, Right’(a)和Right’(b)要么不相交，要么是包含关系；

- 所以可以看出Right()是树形关系，那么记Parent(s)是状态s在树上的祖先。
  即Right(s) ∈ Right(Parent(s))

  举个例子：aaabbaaabd
  Parent(ST(aaab))=Parent(ST(ab))=ST(b)
  容易看出，Max(Parent(s))=Min(s)-1，也就是说，状态s的长度Min(s)-1对应的串的出现次数比Min(s)多了一些

  由此可以证明状态数是线性的，具体证明略

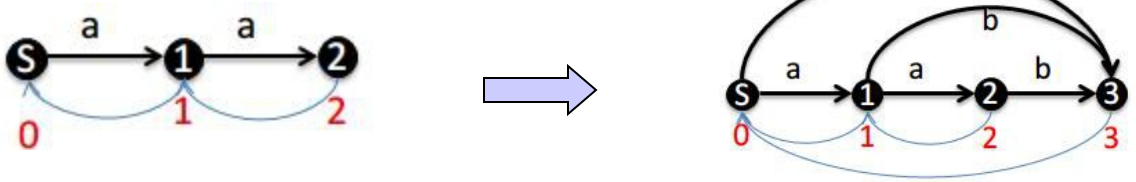
  另外，由于包含关系可知，所有的end就是ST(S)的所有Parent树的祖先

构造算法

- 每次在S末尾添加一个字符x，然后由S的SAM改造成Sx的SAM

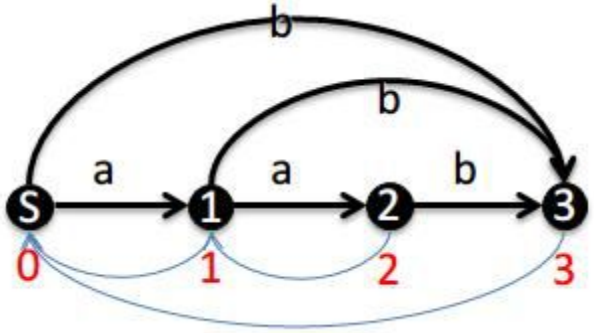
- 设S的长度为len,新建状态np，Right(np)=len+1;
- 若状态s，Right(s)不包含len，那么添加一个字符后，状态s不需要改变。
- 设p=ST(S);由于Right(p)={len},那么可能改变的状态就是p在Parent树上的所有祖先，不妨记为v1=p,v2,v3…vk=root(按深度顺序),那么v1到vk的Right集合是递增的；

- 若ST(vi,x)=NULL，也就是说vi的结束位置之后不存在x,那么由于vi代表的一个串是S的后缀(因为Right(vi)包含len),所以令ST(vi,x)=np，代表在新的自动机上，若一个串转移到了状态vi，再读入一个字符x，转移到的新状态就是np
- 比如原串aa,新加入b,那么ST(2,'b')=ST(1,'b')=ST(s,'b')=np
  (图中蓝色边表示Parent，红色数字表示Max(s))



构造算法

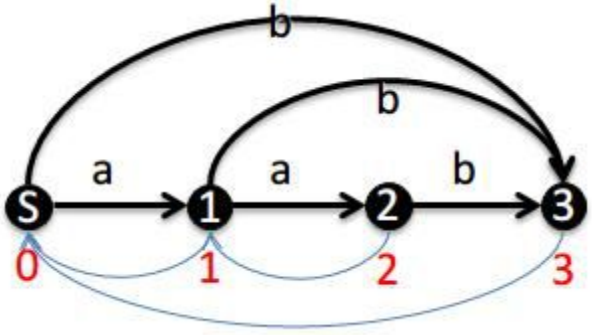
- 若ST(vi,x)=q!=NULL:
- 如原串是aab，它的自动机是这样：



- 现在添加字符b，对于状态3直接连一条到新状态4的b的边,但是对于s怎么办？

构造算法

- s状态的Right集合中，只有那些S[ri]=x的ri仍然是合法的，但是如果直接把ST(s,x)赋为合法的end的话，在图中3就是合法的end了，然而ST(s,”aab”)不是合法后缀却也赋为合法的end了 (此时的串是aabb)



当然，如果Max(vi)+1=Max(q),也就是说不存在另一个串t!=‘x’,使得ST(vi,t)=St(vi,x),那么就可以直接这样赋值为合法了。(就是令Parent(np)=vi，即在Right(vi)中间插入{len+1})

构造算法

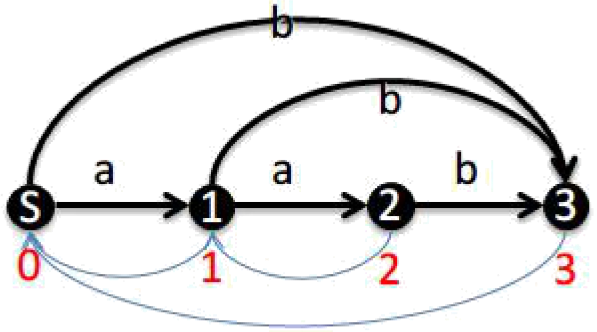
- 如果Max(vi)+1!=Max(q)：

  由于不能直接赋值，那么就把q复制出一个新的节点new，然后令Max(new)=Max(vi)+1，这个时候就可以直接令Right(new)=Right(q)∪{len+1},就是令 Parent(np)=new;可以证明Parent(q)=new；

  随后，所有的Right(vj)包含len的(就是Parent树上的祖先)，并且ST(vj,x)=q的vj状态，令转移ST(vj,x)=new即可。这是因为原来vj可以转移到q而且len∈Right{vj}，那么在新的字符串中，vj可以转移到的状态除了q以外，还多了一个{len+1}

构造算法

- 比如说这张图中，Right(ST(s,b))={3}，然后如果新加了一个字符b的话，Right(ST(s,b))就应该变成{3,4}了



- 再来一个例子：aaabcaabaa,状态vi所代表的字符串用红色标出：aaabcaabaa,这个时候加入一个字符b，原来的q代表的字符串是：aaabcaabaa,现在应该是aaabcaabaab,比原来多了一个{len+1}，所以原来转移到q而且Right集合包含len的状态vj，ST(vj,"b")就应该变成new

具体实现

- 新建节点np,令Max(np)=len+1;
- 设ST(s,S)=last，那么p从last开始，若ST(p,x)=NULL,直接令ST(p,x)=np，直到ST(p,x)!=NULL
- 若不存在这样的p，令Parent(np)=s;结束
- 否则，设ST(p,x)=q,若Max(q)=Max(p)+1,令Parent(np)=q;结束
- 新建节点new=q,令Parent(q)=Parent(np)=new,Max(new)=Max(p)+1,然后再从p开始沿着Parent边，若ST(p,x)=q,就令ST(p,x)=q;

- 代码并不长……

```
void Extend(int C, int Len) {
    int Now = ++Total, P;
    SAM[Now].L = Len;
    for (P = Last; P && !SAM[P].Ch[C]; P = SAM[P].Fail)
        SAM[P].Ch[C] = Now;
    Last = Now;
    if (!P) SAM[Now].Fail = 1;
    else {
        if (SAM[SAM[P].Ch[C]].L == SAM[P].L + 1) SAM[Now].Fail = SAM[P].Ch[C];
        else {
            int New = ++Total, Q = SAM[P].Ch[C];
            SAM[New] = SAM[Q];
            SAM[New].L = SAM[P].L + 1;
            SAM[Q].Fail = SAM[Now].Fail = New;
            for (; P && (SAM[P].Ch[C] == Q); P = SAM[P].Fail)
                SAM[P].Ch[C] = New;
        }
    }
}
```

- 推荐题目：
  - BZOJ2555
  - BZOJ2806
  - BZOJ3172
  - HDU4416